LAB 4 REPORT

Course: CPS633

Section: 6

Group Members:

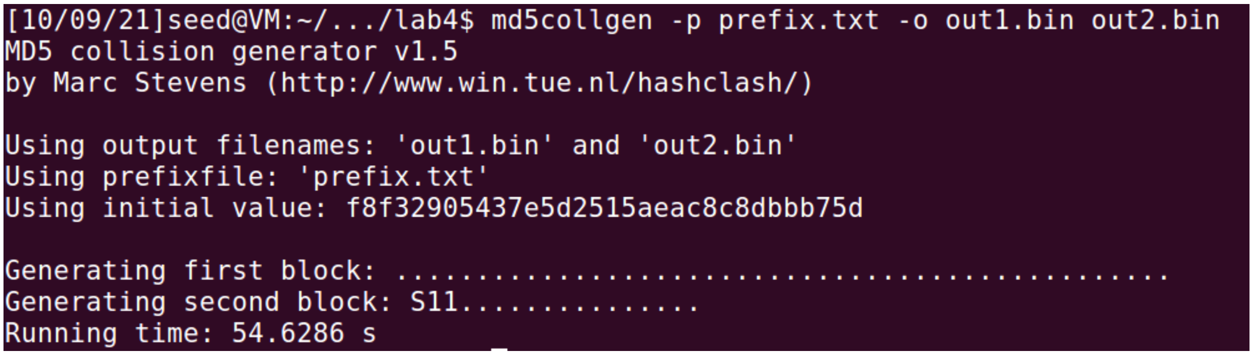
Alishba Aamir, 500974648

Ivan Golovine, 500813431

Fangbo Ren, 500884730

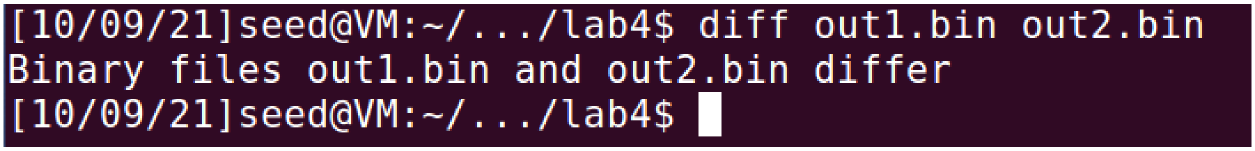
Task 1

We created the prefix.txt file by doing echo “testing” >> prefix.text and ran the md5collgen command to generate the 2 output files.



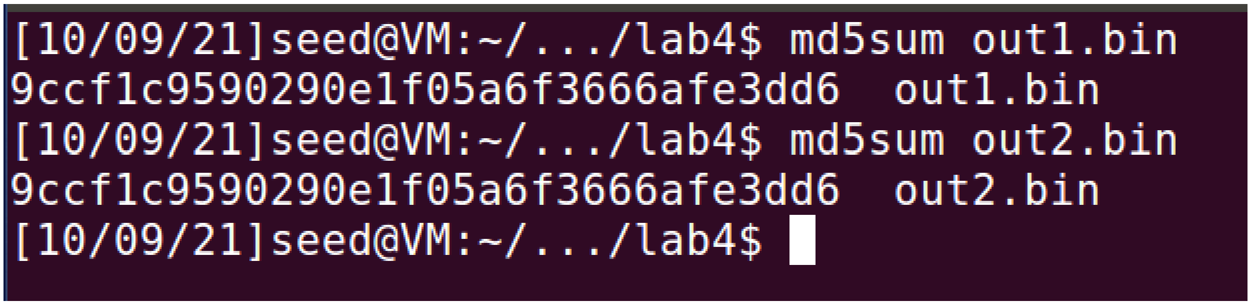
```
[10/09/21]seed@VM:~/.../lab4$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: f8f32905437e5d2515aeac8c8dbbb75d

Generating first block: ................................................
Generating second block: S11...............
Running time: 54.6286 s
```

We then ran the diff out1.bin out2.bin.



```
[10/09/21]seed@VM:~/.../lab4$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[10/09/21]seed@VM:~/.../lab4$ 
```

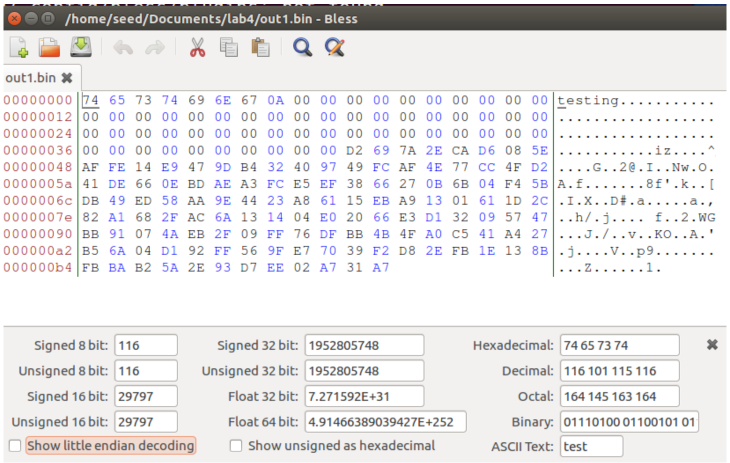
It showed that the binary files differed for both programs. So, we then checked the sum using the md5sum command and it showed that the sum was the same for both files.



```
[10/09/21]seed@VM:~/.../lab4$ md5sum out1.bin
9ccf1c9590290e1f05a6f3666afe3dd6  out1.bin
[10/09/21]seed@VM:~/.../lab4$ md5sum out2.bin
9ccf1c9590290e1f05a6f3666afe3dd6  out2.bin
[10/09/21]seed@VM:~/.../lab4$ 
```

Question 1. If the length of your prefix file is not multiple of 64, what is going to happen?

It is going to be padded with 0’s for it to take up the remaining space in the 64 bytes since MD5 works with 64 byte blocks. By using the xxd or bless command this can be checked.

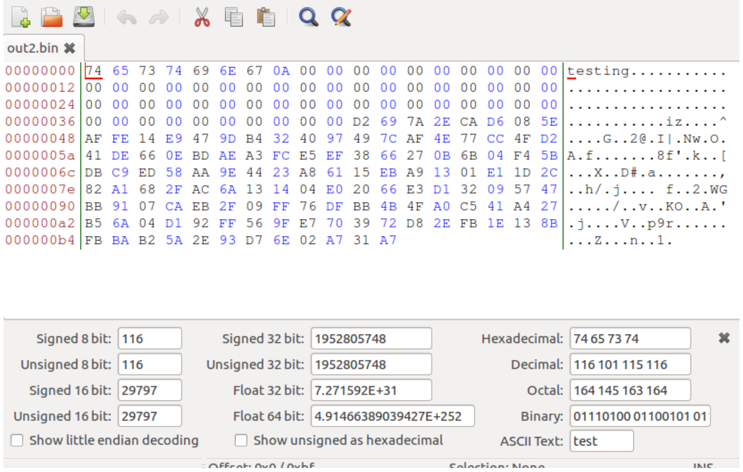


/home/seed/Documents/lab4/out1.bin - Bless

out1.bin

```
00000000 74 65 73 74 69 6E 67 0A 00 00 00 00 00 00 00 00 00 00 testing...........
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000024 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000036 00 00 00 00 00 00 00 00 00 00 D2 69 7A 2E CA D6 08 5E ...........iz....^
00000048 AF FE 14 E9 47 9D B4 32 40 97 49 FC AF 4E 77 CC 4F D2 ....G..2@.I..Nw.O.
0000005a 41 DE 66 0E BD AE A3 FC E5 EF 38 66 27 0B 6B 04 F4 5B A.f.......8f'.k..[
0000006c DB 49 ED 58 AA 9E 44 23 A8 61 15 EB A9 13 01 61 1D 2C .I.X..D#.a.....a.,
0000007e 82 A1 68 2F AC 6A 13 14 04 E0 20 66 E3 D1 32 09 57 47 ..h/.j.... f..2.WG
00000090 BB 91 07 4A EB 2F 09 FF 76 DF BB 4B 4F A0 C5 41 A4 27 ...J./..v..KO..A.'
000000a2 B5 6A 04 D1 92 FF 56 9F E7 70 39 F2 D8 2E FB 1E 13 8B .j....V..p9.......
000000b4 FB BA B2 5A 2E 93 D7 EE 02 A7 31 A7                   ...Z......1.
```

Signed 8 bit: 116
Unsigned 8 bit: 116
Signed 16 bit: 29797
Unsigned 16 bit: 29797
☐ Show little endian decoding
Signed 32 bit: 1952805748
Unsigned 32 bit: 1952805748
Float 32 bit: 7.271592E+31
Float 64 bit: 4.91466389039427E+252
☐ Show unsigned as hexadecimal
Hexadecimal: 74 65 73 74
Decimal: 116 101 115 116
Octal: 164 145 163 164
Binary: 01110100 01100101 01
ASCII Text: test



out2.bin

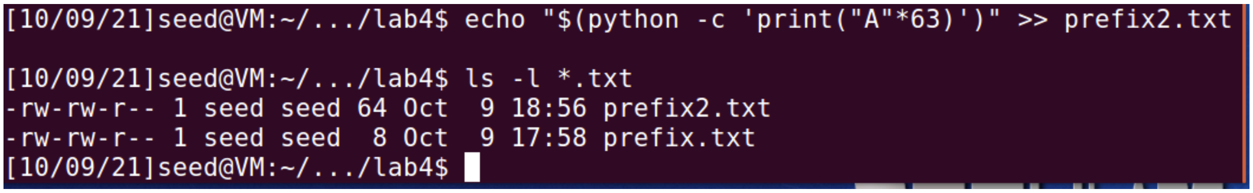
```
00000000 74 65 73 74 69 6E 67 0A 00 00 00 00 00 00 00 00 00 00 testing...........
00000012 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000024 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00000036 00 00 00 00 00 00 00 00 00 00 D2 69 7A 2E CA D6 08 5E ...........iz....^
00000048 AF FE 14 E9 47 9D B4 32 40 97 49 7C AF 4E 77 CC 4F D2 ....G..2@.I|.Nw.O.
0000005a 41 DE 66 0E BD AE A3 FC E5 EF 38 66 27 0B 6B 04 F4 5B A.f.......8f'.k..[
0000006c DB C9 ED 58 AA 9E 44 23 A8 61 15 EB A9 13 01 E1 1D 2C ...X..D#.a.......,
0000007e 82 A1 68 2F AC 6A 13 14 04 E0 20 66 E3 D1 32 09 57 47 ..h/.j.... f..2.WG
00000090 BB 91 07 CA EB 2F 09 FF 76 DF BB 4B 4F A0 C5 41 A4 27 ...../..v..KO..A.'
000000a2 B5 6A 04 D1 92 FF 56 9F E7 70 39 72 D8 2E FB 1E 13 8B .j....V..p9r......
000000b4 FB BA B2 5A 2E 93 D7 6E 02 A7 31 A7                   ...Z...n..1.
```

Signed 8 bit: 116
Unsigned 8 bit: 116
Signed 16 bit: 29797
Unsigned 16 bit: 29797
☐ Show little endian decoding
Signed 32 bit: 1952805748
Unsigned 32 bit: 1952805748
Float 32 bit: 7.271592E+31
Float 64 bit: 4.91466389039427E+252
☐ Show unsigned as hexadecimal
Hexadecimal: 74 65 73 74
Decimal: 116 101 115 116
Octal: 164 145 163 164
Binary: 01110100 01100101 01
ASCII Text: test

Offset: 0x0 / 0xbf
Selection: None
INS

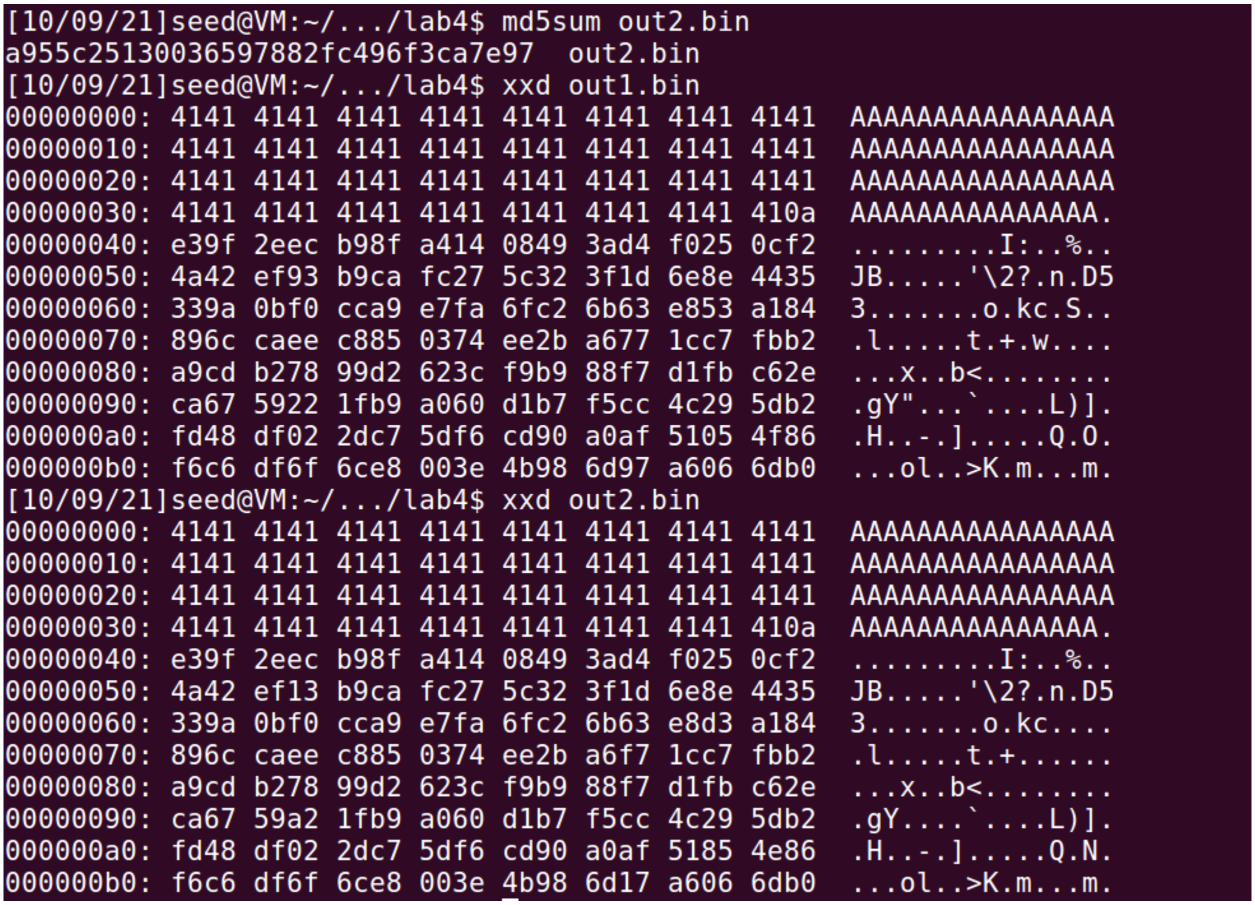
Question 2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

In the image below we check that the prefix2 has 64 bytes.



```
[10/09/21]seed@VM:~/.../lab4$ echo "$(python -c 'print("A"*63)')" >> prefix2.txt

[10/09/21]seed@VM:~/.../lab4$ ls -l *.txt
-rw-rw-r-- 1 seed seed 64 Oct  9 18:56 prefix2.txt
-rw-rw-r-- 1 seed seed  8 Oct  9 17:58 prefix.txt
[10/09/21]seed@VM:~/.../lab4$ 
```

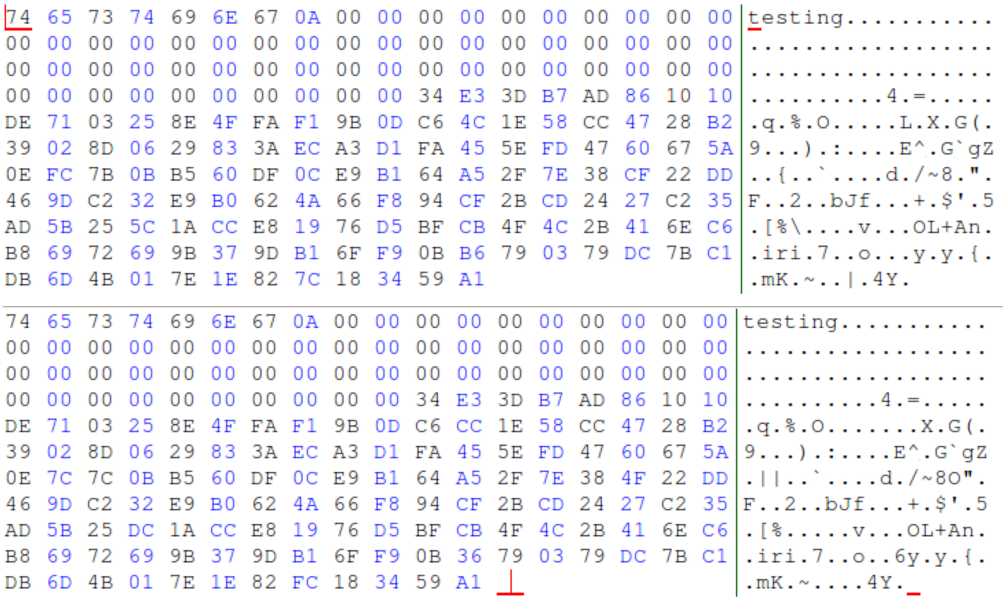
The 0's are no longer there, and everything is taken up by the message being encrypted. So, the output now only has the prefix and p/q which can be seen in the image below.



```
[10/09/21]seed@VM:~/.../lab4$ md5sum out2.bin
a955c25130036597882fc496f3ca7e97  out2.bin
[10/09/21]seed@VM:~/.../lab4$ xxd out1.bin
00000000: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000010: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000020: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000030: 4141 4141 4141 4141 4141 4141 4141 410a  AAAAAAAAAAAAAAA.
00000040: e39f 2eec b98f a414 0849 3ad4 f025 0cf2  .........I:..%..
00000050: 4a42 ef93 b9ca fc27 5c32 3f1d 6e8e 4435  JB.....'\2?.n.D5
00000060: 339a 0bf0 cca9 e7fa 6fc2 6b63 e853 a184  3.......o.kc.S..
00000070: 896c caee c885 0374 ee2b a677 1cc7 fbb2  .l.....t.+.w....
00000080: a9cd b278 99d2 623c f9b9 88f7 d1fb c62e  ...x..b<........
00000090: ca67 5922 1fb9 a060 d1b7 f5cc 4c29 5db2  .gY"...`....L)].
000000a0: fd48 df02 2dc7 5df6 cd90 a0af 5105 4f86  .H..-.].....Q.O.
000000b0: f6c6 df6f 6ce8 003e 4b98 6d97 a606 6db0  ...ol..>K.m...m.
[10/09/21]seed@VM:~/.../lab4$ xxd out2.bin
00000000: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000010: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000020: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000030: 4141 4141 4141 4141 4141 4141 4141 410a  AAAAAAAAAAAAAAA.
00000040: e39f 2eec b98f a414 0849 3ad4 f025 0cf2  .........I:..%..
00000050: 4a42 ef13 b9ca fc27 5c32 3f1d 6e8e 4435  JB.....'\2?.n.D5
00000060: 339a 0bf0 cca9 e7fa 6fc2 6b63 e8d3 a184  3.......o.kc....
00000070: 896c caee c885 0374 ee2b a6f7 1cc7 fbb2  .l.....t.+......
00000080: a9cd b278 99d2 623c f9b9 88f7 d1fb c62e  ...x..b<........
00000090: ca67 59a2 1fb9 a060 d1b7 f5cc 4c29 5db2  .gY....`....L)].
000000a0: fd48 df02 2dc7 5df6 cd90 a0af 5185 4e86  .H..-.].....Q.N.
000000b0: f6c6 df6f 6ce8 003e 4b98 6d17 a606 6db0  ...ol..>K.m...m.
```

Question 3. Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different

No, most of the bytes are the same for both outputs with only a few of them varying, and usually being different when tested multiple times. In the example below the different ones are in positions, 20,46,47,59,84,109, and 123.

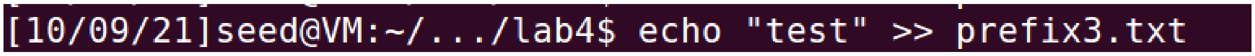


```
74 65 73 74 69 6E 67 0A 00 00 00 00 00 00 00 00 00 00 testing..........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00 00 00 00 00 00 00 00 00 00 34 E3 3D B7 AD 86 10 10 ..........4.=.....
DE 71 03 25 8E 4F FA F1 9B 0D C6 4C 1E 58 CC 47 28 B2 .q.%.O.....L.X.G(.
39 02 8D 06 29 83 3A EC A3 D1 FA 45 5E FD 47 60 67 5A 9...).:....E^.G`gZ
0E FC 7B 0B B5 60 DF 0C E9 B1 64 A5 2F 7E 38 CF 22 DD ..{..`....d./~8.".
46 9D C2 32 E9 B0 62 4A 66 F8 94 CF 2B CD 24 27 C2 35 F..2..bJf...+.$'.5
AD 5B 25 5C 1A CC E8 19 76 D5 BF CB 4F 4C 2B 41 6E C6 .[%\....v...OL+An.
B8 69 72 69 9B 37 9D B1 6F F9 0B B6 79 03 79 DC 7B C1 .iri.7..o...y.y.{.
DB 6D 4B 01 7E 1E 82 7C 18 34 59 A1                   .mK.~..|.4Y.

74 65 73 74 69 6E 67 0A 00 00 00 00 00 00 00 00 00 00 testing..........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
00 00 00 00 00 00 00 00 00 00 34 E3 3D B7 AD 86 10 10 ..........4.=.....
DE 71 03 25 8E 4F FA F1 9B 0D C6 CC 1E 58 CC 47 28 B2 .q.%.O.......X.G(.
39 02 8D 06 29 83 3A EC A3 D1 FA 45 5E FD 47 60 67 5A 9...).:....E^.G`gZ
0E 7C 7C 0B B5 60 DF 0C E9 B1 64 A5 2F 7E 38 4F 22 DD .||..`....d./~8O".
46 9D C2 32 E9 B0 62 4A 66 F8 94 CF 2B CD 24 27 C2 35 F..2..bJf...+.$'.5
AD 5B 25 DC 1A CC E8 19 76 D5 BF CB 4F 4C 2B 41 6E C6 .[%.....v...OL+An.
B8 69 72 69 9B 37 9D B1 6F F9 0B 36 79 03 79 DC 7B C1 .iri.7..o..6y.y.{.
DB 6D 4B 01 7E 1E 82 FC 18 34 59 A1                   .mK.~....4Y.
```

Task 2

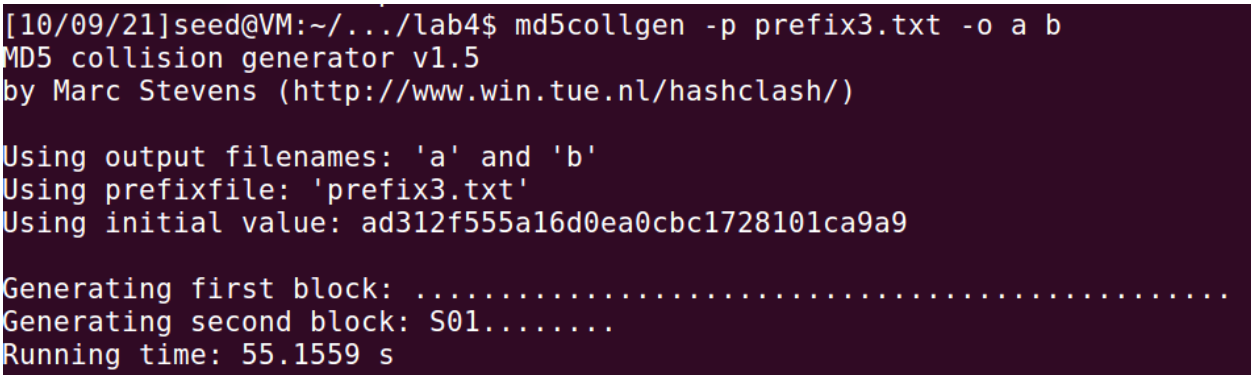
Using the MD5 hash algorithm, if the hash value is the same for both elements then concatenating either a text or binary number to them will produce the same hash for both elements.

First to test this we created a new text file which held the string test.



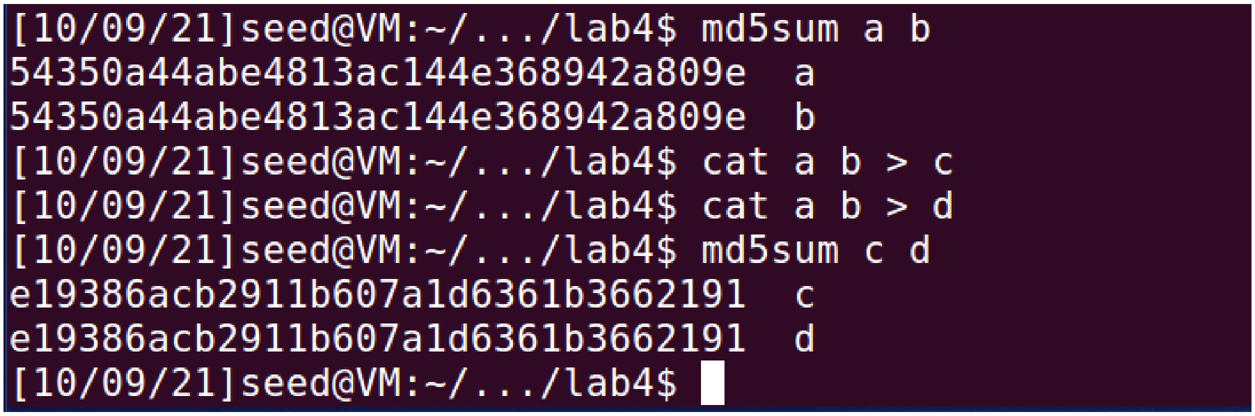
```
[10/09/21]seed@VM:~/.../lab4$ echo "test" >> prefix3.txt
```

Then used the md5collgen to generate the md5 hashes and store them in a and b.



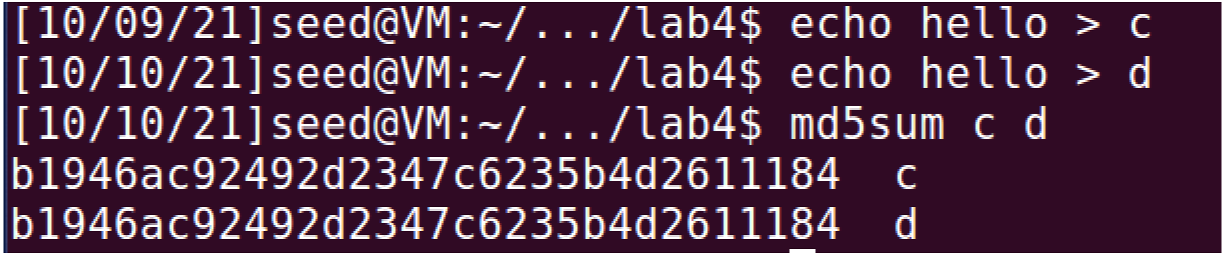
```
[10/09/21]seed@VM:~/.../lab4$ md5collgen -p prefix3.txt -o a b
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'a' and 'b'
Using prefixfile: 'prefix3.txt'
Using initial value: ad312f555a16d0ea0cbc1728101ca9a9

Generating first block: .........................................
Generating second block: S01........
Running time: 55.1559 s
```

We used md5sum to verify that the hashes for both files were the same. Then we concatenated the files using the cat a b > c and cat a b > d and checked md5sum which turned out to be the same.



```
[10/09/21]seed@VM:~/.../lab4$ md5sum a b
54350a44abe4813ac144e368942a809e  a
54350a44abe4813ac144e368942a809e  b
[10/09/21]seed@VM:~/.../lab4$ cat a b > c
[10/09/21]seed@VM:~/.../lab4$ cat a b > d
[10/09/21]seed@VM:~/.../lab4$ md5sum c d
e19386acb2911b607a1d6361b3662191  c
e19386acb2911b607a1d6361b3662191  d
[10/09/21]seed@VM:~/.../lab4$ 
```

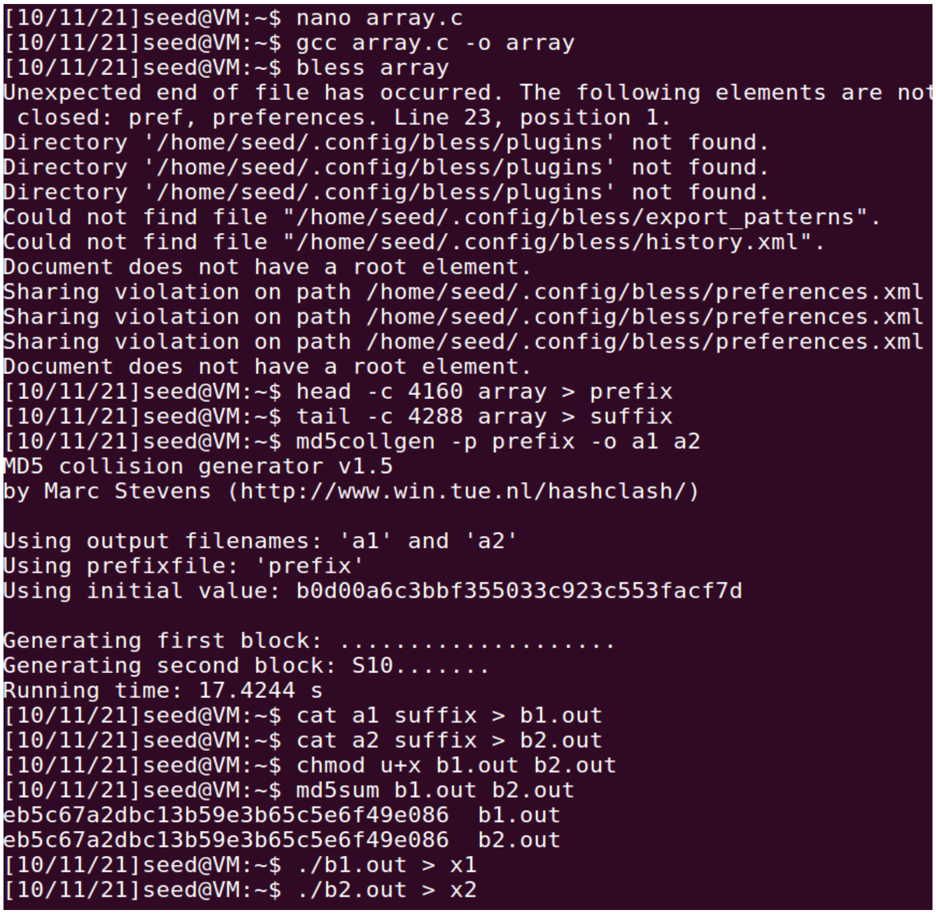
We then tested again by appending c and d with a new string and the result was the same.



```
[10/09/21]seed@VM:~/.../lab4$ echo hello > c
[10/10/21]seed@VM:~/.../lab4$ echo hello > d
[10/10/21]seed@VM:~/.../lab4$ md5sum c d
b1946ac92492d2347c6235b4d2611184  c
b1946ac92492d2347c6235b4d2611184  d
```

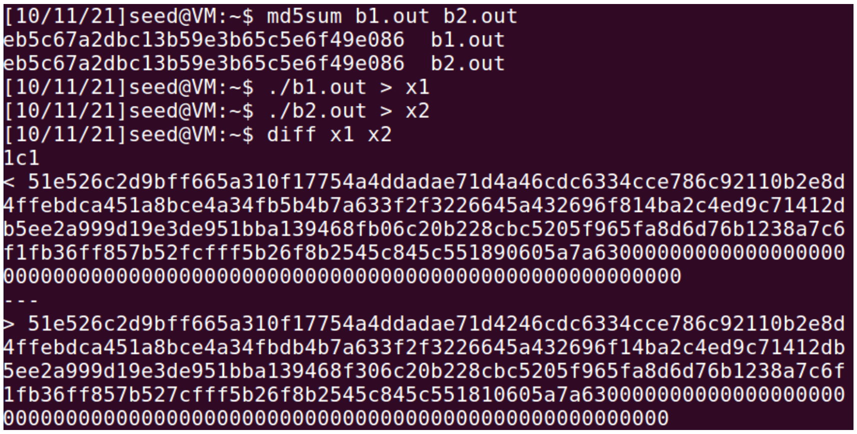
Task 3

First I create the array.c and then use bless to find the location that array is written(0x1040). Then i truncate the prefix and the suffix from the file and generate 2 files with prefix. After that append suffix to them and make it executable.



```
[10/11/21]seed@VM:~$ nano array.c
[10/11/21]seed@VM:~$ gcc array.c -o array
[10/11/21]seed@VM:~$ bless array
Unexpected end of file has occurred. The following elements are not
 closed: pref, preferences. Line 23, position 1.
Directory '/home/seed/.config/bless/plugins' not found.
Directory '/home/seed/.config/bless/plugins' not found.
Directory '/home/seed/.config/bless/plugins' not found.
Could not find file "/home/seed/.config/bless/export_patterns".
Could not find file "/home/seed/.config/bless/history.xml".
Document does not have a root element.
Sharing violation on path /home/seed/.config/bless/preferences.xml
Sharing violation on path /home/seed/.config/bless/preferences.xml
Sharing violation on path /home/seed/.config/bless/preferences.xml
Document does not have a root element.
[10/11/21]seed@VM:~$ head -c 4160 array > prefix
[10/11/21]seed@VM:~$ tail -c 4288 array > suffix
[10/11/21]seed@VM:~$ md5collgen -p prefix -o a1 a2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'a1' and 'a2'
Using prefixfile: 'prefix'
Using initial value: b0d00a6c3bbf355033c923c553facf7d

Generating first block: ...................
Generating second block: S10.......
Running time: 17.4244 s
[10/11/21]seed@VM:~$ cat a1 suffix > b1.out
[10/11/21]seed@VM:~$ cat a2 suffix > b2.out
[10/11/21]seed@VM:~$ chmod u+x b1.out b2.out
[10/11/21]seed@VM:~$ md5sum b1.out b2.out
eb5c67a2dbc13b59e3b65c5e6f49e086  b1.out
eb5c67a2dbc13b59e3b65c5e6f49e086  b2.out
[10/11/21]seed@VM:~$ ./b1.out > x1
[10/11/21]seed@VM:~$ ./b2.out > x2
```

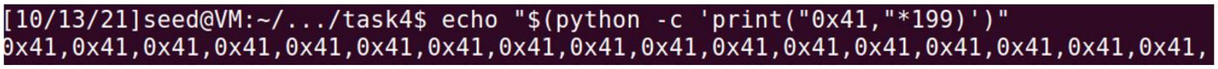
Here use md5sum and see if they have same MD5 hash. Run them and compare the output. It shows that they have same MD5 hash but with different output.



```
[10/11/21]seed@VM:~$ md5sum b1.out b2.out
eb5c67a2dbc13b59e3b65c5e6f49e086  b1.out
eb5c67a2dbc13b59e3b65c5e6f49e086  b2.out
[10/11/21]seed@VM:~$ ./b1.out > x1
[10/11/21]seed@VM:~$ ./b2.out > x2
[10/11/21]seed@VM:~$ diff x1 x2
1c1
< 51e526c2d9bff665a310f17754a4ddadae71d4a46cdc6334cce786c92110b2e8d
4ffebdca451a8bce4a34fb5b4b7a633f2f3226645a432696f814ba2c4ed9c71412d
b5ee2a999d19e3de951bba139468fb06c20b228cbc5205f965fa8d6d76b1238a7c6
f1fb36ff857b52fcfff5b26f8b2545c845c551890605a7a63000000000000000000
000000000000000000000000000000000000000000000000000000
---
> 51e526c2d9bff665a310f17754a4ddadae71d4246cdc6334cce786c92110b2e8d
4ffebdca451a8bce4a34fbdb4b7a633f2f3226645a432696f14ba2c4ed9c71412db
5ee2a999d19e3de951bba139468f306c20b228cbc5205f965fa8d6d76b1238a7c6f
1fb36ff857b527cfff5b26f8b2545c845c551810605a7a630000000000000000000
00000000000000000000000000000000000000000000000000000
```

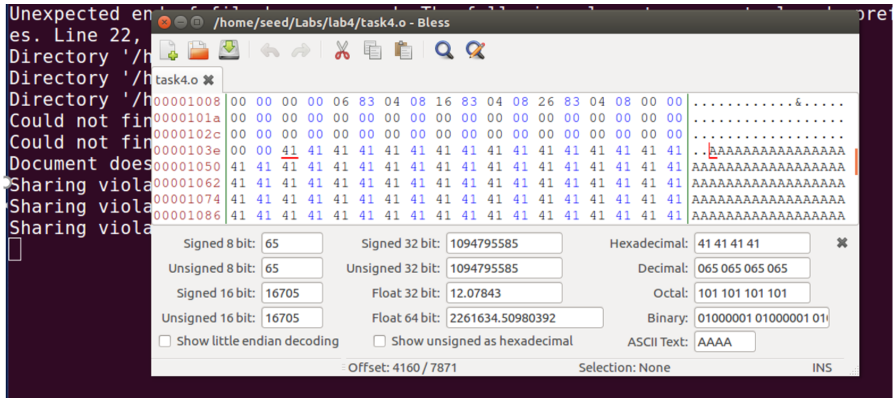
Task 4 :

To start this, we initialized variable x and y and stored the same values in both array. As shown below, we filled the array with “0x41” x 199



```
[10/13/21]seed@VM:~/.../task4$ echo "$(python -c 'print("0x41,"*199)')"
0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
```

After filling out our c program task4.c we compile it and then use the bless tool to view the binary executable file and find the location for the array. As seen below our X array starts at offset location 4160.



```
Unexpected en
es. Line 22,
Directory '/h
Directory '/h
Directory '/h
Could not fin
Could not fin
Document does
Sharing viola
Sharing viola
Sharing viola
```

/home/seed/Labs/lab4/task4.o - Bless

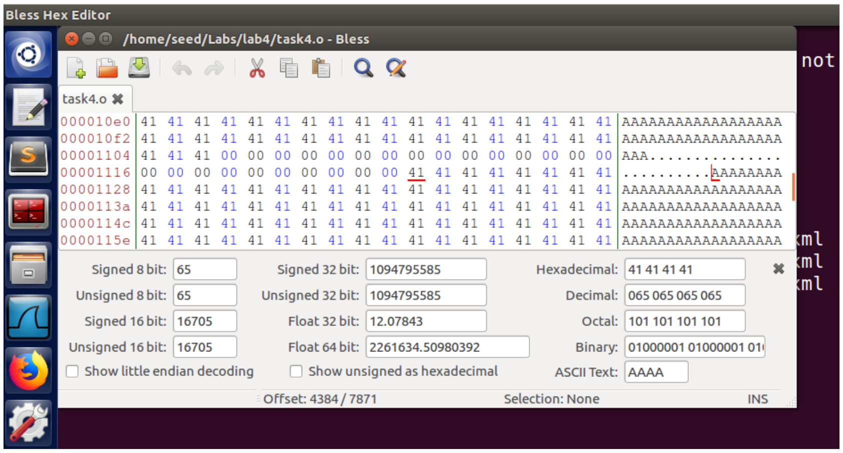
task4.o

```
00001008 00 00 00 00 06 83 04 08 16 83 04 08 26 83 04 08 00 00 ............&.....
0000101a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
0000102c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
0000103e 00 00 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 ..AAAAAAAAAAAAAAAA
00001050 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001062 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001074 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001086 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
```

Signed 8 bit: 65 | Signed 32 bit: 1094795585 | Hexadecimal: 41 41 41 41
Unsigned 8 bit: 65 | Unsigned 32 bit: 1094795585 | Decimal: 065 065 065 065
Signed 16 bit: 16705 | Float 32 bit: 12.07843 | Octal: 101 101 101 101
Unsigned 16 bit: 16705 | Float 64 bit: 2261634.50980392 | Binary: 01000001 01000001 01
☐ Show little endian decoding | ☐ Show unsigned as hexadecimal | ASCII Text: AAAA

Offset: 4160 / 7871 Selection: None INS

The array y starts at offset location 4384.



Bless Hex Editor

/home/seed/Labs/lab4/task4.o - Bless

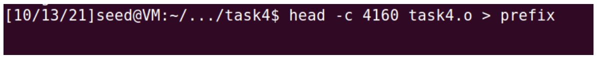
task4.o

```
000010e0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000010f2 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
00001104 41 41 41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AAA...............
00001116 00 00 00 00 00 00 00 00 00 00 41 41 41 41 41 41 41 41 ..........AAAAAAAA
00001128 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0000113a 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0000114c 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0000115e 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
```

Signed 8 bit: 65 | Signed 32 bit: 1094795585 | Hexadecimal: 41 41 41 41
Unsigned 8 bit: 65 | Unsigned 32 bit: 1094795585 | Decimal: 065 065 065 065
Signed 16 bit: 16705 | Float 32 bit: 12.07843 | Octal: 101 101 101 101
Unsigned 16 bit: 16705 | Float 64 bit: 2261634.50980392 | Binary: 01000001 01000001 01
☐ Show little endian decoding | ☐ Show unsigned as hexadecimal | ASCII Text: AAAA

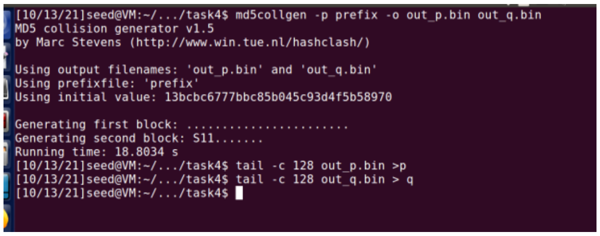
Offset: 4384 / 7871 Selection: None INS

For our prefix, we basically take everything from the start to location 4160 which is the start of the x array. This is done using Head function. (-c stands for bytes)

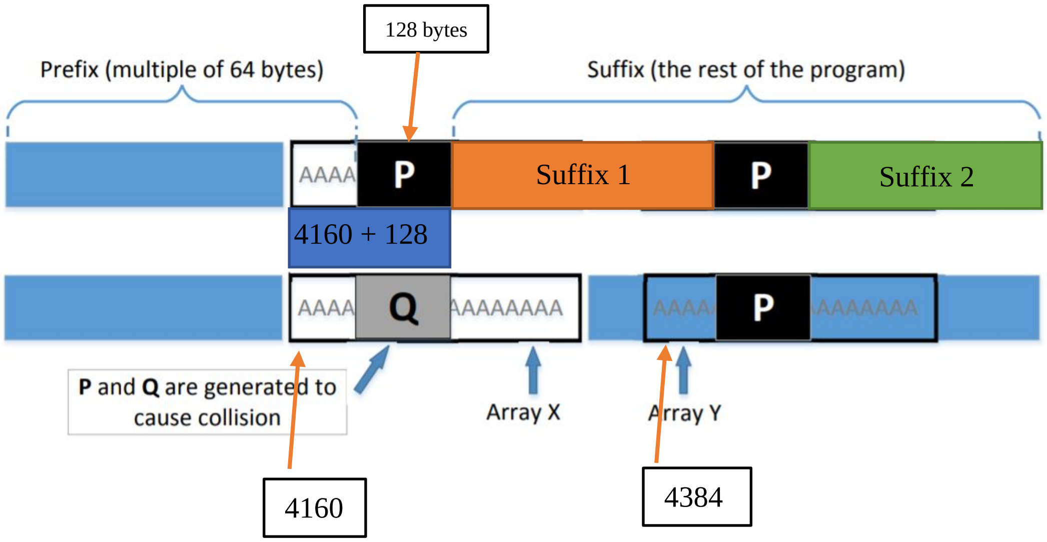


```
[10/13/21]seed@VM:~/.../task4$ head -c 4160 task4.o > prefix
```

Since we will make 2 programs, one that runs the Benign code and one that runs the malicious code, we need both of them to have the sane hash values as explained in the lab manual. To do this, we use md5collgen so that 2 programs have the same hash values. We take the last 128 bytes from the out_p.bin and store it in p and then take the last 128 bytes from the out_q.bin and store it in q

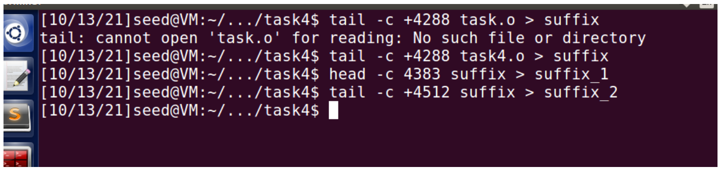


```
[10/13/21]seed@VM:~/.../task4$ md5collgen -p prefix -o out_p.bin out_q.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out_p.bin' and 'out_q.bin'
Using prefixfile: 'prefix'
Using initial value: 13bcbc6777bbc85b045c93d4f5b58970

Generating first block: .......................
Generating second block: S11.......
Running time: 18.8034 s
[10/13/21]seed@VM:~/.../task4$ tail -c 128 out_p.bin >p
[10/13/21]seed@VM:~/.../task4$ tail -c 128 out_q.bin > q
[10/13/21]seed@VM:~/.../task4$ 
```


128 bytes
Prefix (multiple of 64 bytes)
Suffix (the rest of the program)
AAAA
P
Suffix 1
P
Suffix 2
4160 + 128
AAAA
Q
AAAAAAAA
AAAA
P
AAAAAAA
P and Q are generated to cause collision
Array X
Array Y
4160
4384

Now, we use the suffix and divide it in 2 parts. As shown in the picture above, our program should be structured like:
Benign code: Prefix + p + suffix1 + p + suffix2
Malicious code: Prefix + q + suffix1 + p + suffix2
The entire suffix is: 4160 + 128 = 4288 → We use the tail funtion and put everything except the first 4288 bytes of the code from the task4.o file in the suffix file

Suffix1 : 4384 -1 = 4383
Then from suffix, we take from start till 4383 which is the start of the of the Y array. That will be suffix1
Suffix 2 will be 4384+ 128 = 4512. Everything except the first 4512 bytes. We use head and tail for making these files.

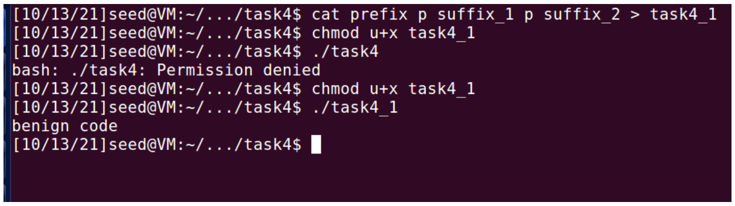


```
[10/13/21]seed@VM:~/.../task4$ tail -c +4288 task.o > suffix
tail: cannot open 'task.o' for reading: No such file or directory
[10/13/21]seed@VM:~/.../task4$ tail -c +4288 task4.o > suffix
[10/13/21]seed@VM:~/.../task4$ head -c 4383 suffix > suffix_1
[10/13/21]seed@VM:~/.../task4$ tail -c +4512 suffix > suffix_2
[10/13/21]seed@VM:~/.../task4$ 
```

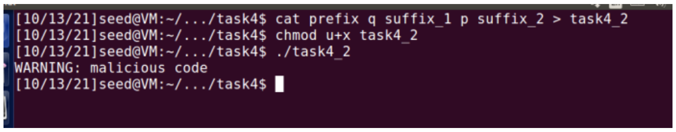
As explained above we will piece together the 2 programs using the files we made before:

Task4_1 = prefix + p + suffix1 + p suffix2

Task4_2 = prefix + p + suffix1 + p suffix2



```
[10/13/21]seed@VM:~/.../task4$ cat prefix p suffix_1 p suffix_2 > task4_1
[10/13/21]seed@VM:~/.../task4$ chmod u+x task4_1
[10/13/21]seed@VM:~/.../task4$ ./task4
bash: ./task4: Permission denied
[10/13/21]seed@VM:~/.../task4$ chmod u+x task4_1
[10/13/21]seed@VM:~/.../task4$ ./task4_1
benign code
[10/13/21]seed@VM:~/.../task4$ 
```



```
[10/13/21]seed@VM:~/.../task4$ cat prefix q suffix_1 p suffix_2 > task4_2
[10/13/21]seed@VM:~/.../task4$ chmod u+x task4_2
[10/13/21]seed@VM:~/.../task4$ ./task4_2
WARNING: malicious code
[10/13/21]seed@VM:~/.../task4$ 
```